# Tetris: The Grand Master 3
# Grade System Mathematical Analysis

Based on Nullpomino Implementation
Source Code Analysis

December 13, 2025

## Contents


**1 Introduction** **3**
1.1 Grade Hierarchy . . . 3

**2 Main Grade Calculation Formula** **3**

**3 Internal Grade System** **3**
3.1 Grade Point Accumulation . . . 3
3.2 Grade Point Decay . . . 4
3.3 Grade Advancement . . . 4

**4 Section Performance System** **4**
4.1 COOL Sections . . . 4
4.2 REGRET Penalties . . . 4

**5 Invisible Roll System** **5**
5.1 Roll Point Accumulation . . . 5
5.2 Roll Completion Bonus . . . 5
5.3 M-Roll Activation . . . 5

**6 Qualified Grade System (Exam Mode)** **5**
6.1 Promotional Exam Grade . . . 5
6.2 Demotion System . . . 6
6.3 Qualified Grade Updates . . . 6
6.4 Exam Triggering . . . 6

**7 Grand Master Requirements** **6**

**8 Lookup Tables** **6**
8.1 Base Point Table . . . 6
8.2 Combo Multiplier Table . . . 7
8.3 Section Time Requirements . . . 7
8.4 Grade Change Requirements . . . 7
8.5 Decay Rate Table . . . 7

**9 Implementation Notes** **8**
9.1 Frame Timing . . . . . 8
9.2 Level Progression . . . . . 8
9.3 Grade Display Constraints . . . . . 8

**10 Conclusion** **8**

**11 References** **9**

**A Grade Enumeration** **9**

**B TypeScript Implementation** **9**
B.1 Type Definitions . . . . . 9
B.2 Constants . . . . . 10
B.3 Utility Functions . . . . . 10
B.4 Grade Point Calculation . . . . . 11
B.5 Grade Decay System . . . . . 11
B.6 Internal Grade Advancement . . . . . 11
B.7 Section COOL/REGRET Processing . . . . . 12
B.8 Roll Points System . . . . . 13
B.9 M-Roll Qualification . . . . . 13
B.10 Qualified Grade System . . . . . 13
B.11 Final Grade Calculation . . . . . 14

# 1 Introduction

Tetris: The Grand Master 3 (TGM3) features one of the most complex grading systems in any puzzle game. This document presents a comprehensive mathematical analysis of the grade calculation system as implemented in Nullpomino, an open-source TGM clone. The grading system consists of multiple interacting subsystems that evaluate player performance across various metrics.

## 1.1 Grade Hierarchy

The TGM3 grade system consists of 33 grades in ascending order:

| **Range** | **Grades** |
|---|---|
| Numeric (0-8) | 9, 8, 7, 6, 5, 4, 3, 2, 1 |
| S-Class (9-17) | S1, S2, S3, S4, S5, S6, S7, S8, S9 |
| Master (18-26) | M1, M2, M3, M4, M5, M6, M7, M8, M9 |
| Master+ (27-31) | M, MK, MV, MO, MM |
| Grand Master (32) | GM |

# 2 Main Grade Calculation Formula

The overall grade at time $t$ is calculated as:

$$\boxed{G(t) = \min\left(32, \max\left(0, G_{\text{base}} + G_{\text{COOL}} - G_{\text{REGRET}} + \lfloor P_{\text{roll}} \rfloor + B_{\text{roll}}\right)\right)} \tag{1}$$

Where:

- $G_{\text{base}}$: Base grade from internal grade system
- $G_{\text{COOL}}$: Bonus grades from COOL sections
- $G_{\text{REGRET}}$: Penalty grades from REGRET sections
- $P_{\text{roll}}$: Points accumulated during invisible roll
- $B_{\text{roll}}$: Bonus for completing the roll

# 3 Internal Grade System

## 3.1 Grade Point Accumulation

The internal grade system tracks points that determine when the base grade increases:

$$P_{\text{internal}} = P_{\text{internal}} + P_{\text{gain}} - D_{\text{decay}} \tag{2}$$

Where the point gain for each line clear is:

$$P_{\text{gain}} = \lfloor B_{\text{points}}[l-1][i] \times M_{\text{combo}}[l-1][c] \times M_{\text{level}} \rfloor \tag{3}$$

With parameters:

- $l \in \{1, 2, 3, 4\}$: Number of lines cleared

- $i = \min(N_{\text{internal}}, 10)$: Internal grade index
- $c = \min(\text{combo} - 1, 9)$: Combo index
- $M_{\text{level}} = 1 + \frac{L}{4000}$: Level multiplier where $L$ is current level

### 3.2 Grade Point Decay

Points decay when not in an active combo:

$$D_{\text{decay}} = \begin{cases} 1 & \text{if frames since last decay} \geq T_{\text{decay}}[N_{\text{internal}}] \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

### 3.3 Grade Advancement

The base grade increases when internal points reach 100:

$$\text{When } P_{\text{internal}} \geq 100 : \begin{cases} P_{\text{internal}} = 0 \\ N_{\text{internal}} = N_{\text{internal}} + 1 \\ G_{\text{base}} = G_{\text{base}} + 1 \text{ if } N_{\text{internal}} \geq T_{\text{change}}[G_{\text{base}}] \end{cases} \tag{5}$$

## 4 Section Performance System

### 4.1 COOL Sections

A section receives a COOL bonus if completed within the time limit:

$$\text{COOL}[s] = \begin{cases} 1 & \text{if } t_s \leq T_{\text{COOL}}[s] \wedge (\text{first COOL} \vee t_s \leq t_{s-1} + 120) \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Total COOL bonus:

$$G_{\text{COOL}} = \sum_{s=0}^{9} \text{COOL}[s] \tag{7}$$

### 4.2 REGRET Penalties

Sections completed too slowly incur penalties:

$$\text{REGRET}[s] = \begin{cases} 1 & \text{if } t_s > T_{\text{REGRET}}[s] \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

Total REGRET penalty:

$$G_{\text{REGRET}} = \sum_{s=0}^{9} \text{REGRET}[s] \tag{9}$$

# 5 Invisible Roll System

## 5.1 Roll Point Accumulation

During the 3238-frame invisible roll at level 999, points are earned for line clears:

$$p_k = \begin{cases} 0.1 & \text{if single line} \\ 0.2 & \text{if double} \\ 0.3 & \text{if triple} \\ 1.0 & \text{if tetris} \end{cases} \tag{10}$$

Total roll points:

$$P_{\text{roll}} = \sum_{k=1}^{n} p_k \tag{11}$$

## 5.2 Roll Completion Bonus

Surviving the entire roll grants additional points:

$$B_{\text{roll}} = \begin{cases} 1.6 & \text{if M-roll (vanish) survived} \\ 0.5 & \text{if regular roll survived} \\ 0 & \text{if roll failed} \end{cases} \tag{12}$$

## 5.3 M-Roll Activation

The vanishing M-roll activates if:

$$\text{M-roll} = \begin{cases} \text{true} & \text{if } G_{\text{pre-roll}} \geq 24 \land C_{\text{total}} \geq 9 \\ \text{false} & \text{otherwise} \end{cases} \tag{13}$$

# 6 Qualified Grade System (Exam Mode)

## 6.1 Promotional Exam Grade

The promotional exam grade is determined by recent performance history:

$$E_p = \max \left\{ g \in \mathcal{G} : \sum_{j=0}^{6} \mathbb{1}[H_j \geq g] > 3 \right\} \tag{14}$$

Where:

- $\mathcal{G} = \{0, 1, ..., 32\}$: Set of all grades
- $H_j$: Grade achieved in the $j$-th most recent game
- $\mathbb{1}[\cdot]$: Indicator function

This finds the highest grade achieved in more than half of the last 7 games.

### 6.2 Demotion System

Demotion points accumulate for poor performance:

$$D_{n+1} = D_n + \max(0, Q_n - G_{\text{final}} - 7) \tag{15}$$

A demotion exam triggers when $D \geq 30$.

### 6.3 Qualified Grade Updates

The qualified grade updates based on exam results:

$$Q_{n+1} = \begin{cases} E_p & \text{if promotion exam passed: } G_{\text{final}} \geq E_p \\ Q_n - 1 & \text{if demotion exam failed: } D \geq 30 \wedge G_{\text{final}} < Q_n \\ Q_n & \text{otherwise (no change)} \end{cases} \tag{16}$$

### 6.4 Exam Triggering

Exams occur with probability:

$$P(\text{exam}) = \frac{1}{3} \text{ when starting a new game} \tag{17}$$

Subject to conditions: 20G mode OFF, Big mode OFF, Exam mode enabled

## 7 Grand Master Requirements

Achieving Grand Master (grade 32) requires satisfying all conditions:

$$\text{GM} = \begin{cases} \text{true} & \text{if all conditions met:} \\ & \bullet\ G_{\text{pre-roll}} \geq 31 \text{ (Master M or higher)} \\ & \bullet\ C_{\text{total}} \geq 9 \text{ (9+ COOL sections)} \\ & \bullet\ \text{Survived entire roll} \\ & \bullet\ Q_n \geq 31 \text{ (if exam mode enabled)} \\ \text{false} & \text{otherwise} \end{cases} \tag{18}$$

## 8 Lookup Tables

### 8.1 Base Point Table

Table 1: Base points by lines cleared and internal grade

| **Lines** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10+** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 10 | 10 | 10 | 10 | 10 | 5 | 5 | 5 | 5 | 5 | 2 |
| 2 | 20 | 20 | 20 | 15 | 15 | 15 | 10 | 10 | 10 | 10 | 12 |
| 3 | 40 | 30 | 30 | 30 | 20 | 20 | 20 | 15 | 15 | 15 | 13 |
| 4 | 50 | 40 | 40 | 40 | 40 | 30 | 30 | 30 | 30 | 30 | 30 |

## 8.2 Combo Multiplier Table

Table 2: Combo multipliers by lines and combo count

| Lines | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10+ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| 2 | 1.0 | 1.2 | 1.2 | 1.4 | 1.4 | 1.4 | 1.4 | 1.5 | 1.5 | 2.0 |
| 3 | 1.0 | 1.4 | 1.5 | 1.6 | 1.7 | 1.8 | 1.9 | 2.0 | 2.1 | 2.5 |
| 4 | 1.0 | 1.5 | 1.8 | 2.0 | 2.2 | 2.3 | 2.4 | 2.5 | 2.6 | 3.0 |

## 8.3 Section Time Requirements

Table 3: Section time requirements (in frames, 60fps)

| Section | COOL Time | REGRET Time |
|---|---|---|
| 0 (0-99) | 3120 (52s) | 5400 (90s) |
| 1 (100-199) | 3120 (52s) | 4500 (75s) |
| 2 (200-299) | 2940 (49s) | 4500 (75s) |
| 3 (300-399) | 2700 (45s) | 4080 (68s) |
| 4 (400-499) | 2700 (45s) | 3600 (60s) |
| 5 (500-599) | 2520 (42s) | 3600 (60s) |
| 6 (600-699) | 2520 (42s) | 3000 (50s) |
| 7 (700-799) | 2280 (38s) | 3000 (50s) |
| 8 (800-899) | 2280 (38s) | 3000 (50s) |
| 9 (900-999) | $\infty$ | 3000 (50s) |

## 8.4 Grade Change Requirements

The internal grade required to advance to each grade:

Table 4: Internal grade requirements for grade advancement

| Grade | Required | Grade | Required | Grade | Required |
|---|---|---|---|---|---|
| 9 → 8 | 1 | S3 → S4 | 12 | M7 → M8 | 25 |
| 8 → 7 | 2 | S4 → S5 | 15 | M8 → M9 | 27 |
| 7 → 6 | 3 | S5 → S6 | 18 | M9 → M | 29 |
| 6 → 5 | 4 | S6 → S7 | 19 | M → MK | 31 |
| 5 → 4 | 5 | S7 → S8 | 20 | MK → MV | – |
| 4 → 3 | 7 | S8 → S9 | 23 | MV → MO | – |
| 3 → 2 | 9 | S9 → M1 | – | MO → MM | – |
| 2 → 1 | – | ... | ... | MM → GM | Special |

## 8.5 Decay Rate Table

Frames between point decay by internal grade:

Table 5: Decay rate (frames between -1 point)

| **Internal** | 0 | 1 | 2 | 3-4 | 5-7 | 8-10 | 11-12 | 13-14 |
|---|---|---|---|---|---|---|---|---|
| **Frames** | 125 | 80 | 80 | 50 | 45 | 40 | 30 | 30 |
| **Internal** | 15-19 | 20-24 | 25-29 | 30-31 | | | | |
| **Frames** | 20 | 15 | 15 | 10 | | | | |

# 9 Implementation Notes

## 9.1 Frame Timing

All timing in TGM3 is measured in frames at 60fps. To convert to seconds: seconds = $\frac{\text{frames}}{60}$

## 9.2 Level Progression

The game progresses from level 0 to 999, divided into 10 sections of 100 levels each. Special mechanics activate at certain levels:

- Level 500: Torikan checkpoint (must reach within 7:00 for MM grade or higher)
- Level 900: Speed significantly increases
- Level 999: Invisible roll begins

## 9.3 Grade Display Constraints

In exam mode, the displayed grade is constrained:

$$G_{\text{display}} = \begin{cases} \min(31, G(t)) & \text{if exam mode enabled} \wedge Q_n < 32 \\ G(t) & \text{otherwise} \end{cases} \tag{19}$$

This prevents displaying GM grade unless the player has a qualified grade of at least MM.

# 10 Conclusion

The TGM3 grade system represents one of the most sophisticated performance evaluation systems in gaming. It combines multiple subsystems that evaluate:

1. **Consistency**: Through the internal grade point system with decay
2. **Speed**: Via COOL section requirements and REGRET penalties
3. **Skill under pressure**: Through the invisible roll system
4. **Long-term performance**: Via the qualified grade system
5. **Peak performance**: Through Grand Master requirements

This mathematical framework creates a grading system that is both challenging and fair, rewarding players who demonstrate mastery across all aspects of the game while providing clear goals for improvement.

# 11 References

- Nullpomino source code: `GradeMania3Mode.java`
- Implementation version: 2
- Analysis date: December 13, 2025

# A Grade Enumeration

Complete mapping of grade indices to display names:

| Index | Name | Index | Name | Index | Name |
|---|---|---|---|---|---|
| 0 | 9 | 11 | S3 | 22 | M4 |
| 1 | 8 | 12 | S4 | 23 | M5 |
| 2 | 7 | 13 | S5 | 24 | M6 |
| 3 | 6 | 14 | S6 | 25 | M7 |
| 4 | 5 | 15 | S7 | 26 | M8 |
| 5 | 4 | 16 | S8 | 27 | M9 |
| 6 | 3 | 17 | S9 | 28 | M |
| 7 | 2 | 18 | M1 | 29 | MK |
| 8 | 1 | 19 | M2 | 30 | MV |
| 9 | S1 | 20 | M3 | 31 | MO |
| 10 | S2 | 21 | MM | 32 | GM |

# B TypeScript Implementation

The following TypeScript implementation demonstrates the complete grade calculation system using a purely functional approach with arrow function expressions.

## B.1 Type Definitions

```
interface GradeState {
  readonly grade: number;
  readonly internalGrade: number;
  readonly internalPoints: number;
  readonly decayFrames: number;
  readonly coolCount: number;
}

interface SectionResult {
  readonly time: number;
  readonly cool: boolean;
  readonly regret: boolean;
}

interface RollState {
  readonly active: boolean;
  readonly points: number;
  readonly completed: boolean;
  readonly mRoll: boolean;
}

interface GameState {
  readonly level: number;
```

```
  readonly combo: number;
  readonly timerActive: boolean;
  readonly sections: readonly SectionResult[];
  readonly gradeState: GradeState;
  readonly rollState: RollState;
  readonly examMode: boolean;
  readonly qualifiedGrade: number;
  readonly gradeHistory: readonly number[];
}

interface QualifiedGradeResult {
  readonly qualifiedGrade: number;
  readonly demotionPoints: number;
}

interface SectionProcessResult {
  readonly gradeState: GradeState;
  readonly cool: boolean;
  readonly regret: boolean;
}

interface RollProcessResult {
  readonly rollState: RollState;
  readonly gradeState: GradeState;
}
```

## B.2 Constants

```
const GRADE_NAMES = [
  "9", "8", "7", "6", "5", "4", "3", "2", "1",
  "S1", "S2", "S3", "S4", "S5", "S6", "S7", "S8", "S9",
  "M1", "M2", "M3", "M4", "M5", "M6", "M7", "M8", "M9",
  "M", "MK", "MV", "MO", "MM", "GM"
] as const;

const COOL_TIMES = [3120, 3120, 2940, 2700, 2700, 2520, 2520, 2280, 2280, Infinity] as
    const;
const REGRET_TIMES = [5400, 4500, 4500, 4080, 3600, 3600, 3000, 3000, 3000, 3000] as
    const;

const GRADE_POINTS = [
  [10, 10, 10, 10, 10, 5, 5, 5, 5, 5, 2],
  [20, 20, 20, 15, 15, 15, 10, 10, 10, 10, 12],
  [40, 30, 30, 30, 20, 20, 20, 15, 15, 15, 13],
  [50, 40, 40, 40, 40, 30, 30, 30, 30, 30, 30]
] as const;

const COMBO_MULTIPLIERS = [
  [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0],
  [1.0, 1.2, 1.2, 1.4, 1.4, 1.4, 1.4, 1.5, 1.5, 2.0],
  [1.0, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.5],
  [1.0, 1.5, 1.8, 2.0, 2.2, 2.3, 2.4, 2.5, 2.6, 3.0]
] as const;

const GRADE_CHANGE_THRESHOLDS = [1, 2, 3, 4, 5, 7, 9, 12, 15, 18, 19, 20, 23, 25, 27,
    29, 31, -1] as const;

const DECAY_RATES = [
  125, 80, 80, 50, 45, 45, 45, 40, 40, 40, 40, 40,
  30, 30, 30, 20, 20, 20, 20, 20, 15, 15, 15, 15,
  15, 15, 15, 15, 15, 15, 10, 10
] as const;

const ROLL_POINTS_BY_LINES = [0.1, 0.2, 0.3, 1.0] as const;

const GRADE_HISTORY_SIZE = 7;
```

## B.3 Utility Functions

```
const clamp = (min: number, max: number) => (value: number): number =>
  Math.max(min, Math.min(max, value));

const clampGrade = clamp(0, 32);
const clampDisplayGrade = clamp(0, 31);

const range = (n: number): number[] =>
  Array.from({ length: n }, (_, i) => i);

const countWhere = <T>(predicate: (item: T) => boolean) => (arr: readonly T[]): number
    =>
  arr.filter(predicate).length;

const findLastIndex = <T>(predicate: (item: T) => boolean) => (arr: readonly T[]):
    number =>
  arr.reduceRight((found, item, idx) => found !== -1 ? found : (predicate(item) ? idx :
      -1), -1);
```

## B.4 Grade Point Calculation

```
const calculatePointGain = (
  linesCleared: number,
  internalGrade: number,
  combo: number,
  level: number
): number =>
  linesCleared < 1 || linesCleared > 4
    ? 0
    : Math.floor(
        GRADE_POINTS[linesCleared - 1][Math.min(internalGrade, 10)] *
        COMBO_MULTIPLIERS[linesCleared - 1][clamp(0, 9)(combo - 1)] *
        (1 + level * 0.00025)
      );
```

## B.5 Grade Decay System

```
const shouldDecay = (
  timerActive: boolean,
  points: number,
  combo: number,
  pieceLocking: boolean
): boolean =>
  timerActive && points > 0 && combo <= 0 && !pieceLocking;

const getDecayThreshold = (internalGrade: number): number =>
  DECAY_RATES[Math.min(internalGrade, DECAY_RATES.length - 1)];

const processDecay = (
  state: GradeState,
  timerActive: boolean,
  combo: number,
  pieceLocking: boolean
): GradeState =>
  !shouldDecay(timerActive, state.internalPoints, combo, pieceLocking)
    ? state
    : state.decayFrames + 1 >= getDecayThreshold(state.internalGrade)
      ? { ...state, decayFrames: 0, internalPoints: state.internalPoints - 1 }
      : { ...state, decayFrames: state.decayFrames + 1 };
```

## B.6 Internal Grade Advancement

```
const shouldAdvanceGrade = (internalGrade: number, currentGrade: number): boolean =>
  GRADE_CHANGE_THRESHOLDS[currentGrade] !== -1 &&
  internalGrade >= GRADE_CHANGE_THRESHOLDS[currentGrade];
```

```
const processPointOverflow = (state: GradeState): GradeState =>
  state.internalPoints < 100
    ? state
    : {
        ...state,
        internalPoints: 0,
        decayFrames: 0,
        internalGrade: state.internalGrade + 1,
        grade: shouldAdvanceGrade(state.internalGrade + 1, state.grade)
          ? clampDisplayGrade(state.grade + 1)
          : state.grade
      };

const processLinesClear = (
  state: GradeState,
  linesCleared: number,
  combo: number,
  level: number
): GradeState =>
  linesCleared === 0
    ? state
    : processPointOverflow({
        ...state,
        internalPoints: state.internalPoints +
          calculatePointGain(linesCleared, state.internalGrade, combo, level)
      });
```

## B.7 Section COOL/REGRET Processing

```
const isCool = (
  sectionIndex: number,
  sectionTime: number,
  previousSectionTime: number,
  previousCool: boolean
): boolean =>
  sectionTime <= COOL_TIMES[sectionIndex] &&
  (!previousCool || sectionTime <= previousSectionTime + 120);

const isRegret = (sectionIndex: number, sectionTime: number): boolean =>
  sectionTime > REGRET_TIMES[sectionIndex];

const applyCool = (state: GradeState): GradeState => ({
  ...state,
  coolCount: state.coolCount + 1,
  grade: clampDisplayGrade(state.grade + 1)
});

const applyRegret = (state: GradeState): GradeState => ({
  ...state,
  coolCount: Math.max(state.coolCount - 1, 0),
  grade: Math.max(state.grade - 1, 0)
});

const processSectionComplete = (
  gradeState: GradeState,
  sectionIndex: number,
  sectionTime: number,
  previousSectionTime: number,
  previousCool: boolean
): SectionProcessResult => {
  const cool = isCool(sectionIndex, sectionTime, previousSectionTime, previousCool);
  const regret = isRegret(sectionIndex, sectionTime);

  const stateAfterCool = cool ? applyCool(gradeState) : gradeState;
  const finalState = regret ? applyRegret(stateAfterCool) : stateAfterCool;

  return { gradeState: finalState, cool, regret };
};

const calculateSectionBonuses = (sections: readonly SectionResult[]): number =>
```

```
  countWhere((s: SectionResult) => s.cool)(sections) -
  countWhere((s: SectionResult) => s.regret)(sections);
```

## B.8 Roll Points System

```
const getRollPoints = (linesCleared: number): number =>
  linesCleared >= 1 && linesCleared <= 4
    ? ROLL_POINTS_BY_LINES[linesCleared - 1]
    : 0;

const convertRollPointsToGrade = (
  points: number,
  grade: number,
  maxGrade: number
): { points: number; grade: number } =>
  points < 1.0 || grade >= maxGrade
    ? { points, grade }
    : convertRollPointsToGrade(points - 1.0, grade + 1, maxGrade);

const processRollLineClear = (
  rollState: RollState,
  gradeState: GradeState,
  linesCleared: number
): RollProcessResult =>
  !rollState.active || linesCleared < 1
    ? { rollState, gradeState }
    : (() => {
        const newPoints = rollState.points + getRollPoints(linesCleared);
        const converted = convertRollPointsToGrade(newPoints, gradeState.grade, 31);
        return {
          rollState: { ...rollState, points: converted.points },
          gradeState: { ...gradeState, grade: converted.grade }
        };
      })();

const processRollComplete = (rollState: RollState, gradeState: GradeState): GradeState
    =>
  !rollState.completed
    ? gradeState
    : (() => {
        const bonus = rollState.mRoll ? 1.6 : 0.5;
        const converted = convertRollPointsToGrade(
          rollState.points + bonus,
          gradeState.grade,
          32
        );
        return { ...gradeState, grade: converted.grade };
      })();
```

## B.9 M-Roll Qualification

```
const checkMRollQualification = (gradeState: GradeState): boolean =>
  gradeState.grade >= 24 && gradeState.coolCount >= 9;
```

## B.10 Qualified Grade System

```
const isHistoryComplete = (history: readonly number[]): boolean =>
  !history.some(g => g === -1);

const countGamesAtOrAbove = (grade: number) => (history: readonly number[]): number =>
  countWhere((g: number) => g >= grade)(history);

const calculatePromotionalExam = (gradeHistory: readonly number[]): number =>
  !isHistoryComplete(gradeHistory)
    ? 0
```

```
    : range(GRADE_NAMES.length)
        .reverse()
        .find(grade => countGamesAtOrAbove(grade)(gradeHistory) > 3) ?? 0;

const calculateDemotionPoints = (
  currentPoints: number,
  qualifiedGrade: number,
  achievedGrade: number
): number =>
  achievedGrade < qualifiedGrade - 7
    ? currentPoints + (qualifiedGrade - achievedGrade - 7)
    : currentPoints;

const updateQualifiedGrade = (
  qualifiedGrade: number,
  achievedGrade: number,
  promotionalExam: number,
  demotionPoints: number,
  promotionFlag: boolean,
  demotionFlag: boolean
): QualifiedGradeResult =>
  promotionFlag && achievedGrade >= promotionalExam
    ? { qualifiedGrade: promotionalExam, demotionPoints: 0 }
    : demotionFlag && achievedGrade < qualifiedGrade
      ? { qualifiedGrade: Math.max(qualifiedGrade - 1, 0), demotionPoints }
      : {
          qualifiedGrade,
          demotionPoints: calculateDemotionPoints(demotionPoints, qualifiedGrade,
              achievedGrade)
        };
```

## B.11 Final Grade Calculation

```
const applyExamConstraint = (
  grade: number,
  examMode: boolean,
  qualifiedGrade: number
): number =>
  examMode && qualifiedGrade < 32
    ? clampDisplayGrade(grade)
    : grade;

const calculateFinalGrade = (gameState: GameState): number =>
  applyExamConstraint(
    clampGrade(
      processRollComplete(gameState.rollState, gameState.gradeState).grade
    ),
    gameState.examMode,
    gameState.qualifiedGrade
  );

const getGradeName = (grade: number): string =>
  grade >= 0 && grade < GRADE_NAMES.length
    ? GRADE_NAMES[grade]
    : "N/A";

// Compose the complete grade from all components
const computeTotalGrade = (
  baseGrade: number,
  sections: readonly SectionResult[],
  rollPoints: number,
  rollBonus: number
): number =>
  clampGrade(
    baseGrade +
    calculateSectionBonuses(sections) +
    Math.floor(rollPoints) +
    Math.floor(rollBonus)
  );
```